Copy No._____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

XML Based Manipulation of Codes Exchanged Between Data Systems

Anthony W. Isenor

Defence R&D Canada

Technical Memorandum
DRDC Atlantic TM 2003-132
August 2003

Canada

Copy No:

XML Based Manipulation of Codes Exchanged Between Data Systems

Anthony W. Isenor

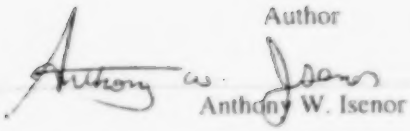
Defence R&D Canada – Atlantic

Technical Memorandum

DRDC Atlantic TM 2003-132

August 2003

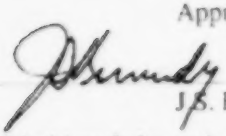
Author



Anthony W. Isenor

Defence Scientist

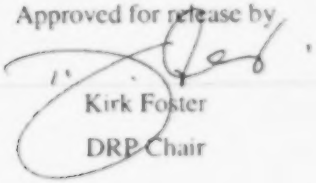
Approved by



J.S. Kennedy

Head, Maritime Information and Combat Systems Section

Approved for release by



Kirk Foster

DRP Chair

© Her Majesty the Queen as represented by the Minister of National Defence, 2003

© Sa majesté la reine, représentée par le ministre de la Défense nationale, 2003

106-03088/

Abstract

Codes are an abbreviated form of information that are often found in many databases. Codes provide the database with a compaction of information and concise definition of data values. For most databases, the codes are developed specifically for the local database and processing system. In a data-sharing environment, the specific code development results in different database systems referring to similar data using different codes. The conversion of codes is required when these systems share data. The eXtensible Markup Language (XML) is used to construct a code mapping between two systems and eXtensible Stylesheet Language Transformations (XSLT) are used to convert the codes in the XML documents.

Résumé

Les codes représentent une forme abrégée de l'information et on les retrouve souvent dans de nombreuses bases de données. Ils permettent de définir les valeurs des données dans une forme concise et comprimée. Dans la plupart des cas, les codes sont élaborés spécifiquement pour la base de données et le système de traitement concernés. Dans un environnement de partage des données, l'existence de différents codes fait que différents systèmes de base de données renvoient aux mêmes données en utilisant des codes différents. Lorsque ces systèmes se partagent des données, les codes doivent être convertis. XML (eXtensible Markup Language) est utilisé pour construire une correspondance entre les codes de deux systèmes, et XSLT (eXtensible Stylesheet Language Transformations) est utilisé pour convertir les codes présents dans les documents XML.

This page intentionally left blank.

Executive summary

Background

Data and information are often stored in a database in coded form. The codes, which are letters, numbers, or words, provide a mechanism for brevity in the database. Typically, autonomous databases have unique codes that are developed based on the specific problem for which the database was designed, and based on the experience of the data modeller.

When information is exchanged between databases, the coded information in one database may not be directly transferable to the other database. Similar, but different codes may exist in the two databases, or existing codes for variables may have different units. In these cases, a mapping between codes must take place.

Mappings result in codes in a particular database being associated with codes in a different database. Mappings are often tedious and laborious, involving experts in the topic area of the code set.

One database of particular interest to the military research community is the Land Command and Control Information Exchange Data Model (LC2IEDM). Researchers are developing this data model to create a common context between disparate systems. In this role, the extensive code set utilized by the LC2IEDM is of particular interest due to the required code mappings between the LC2IEDM and these systems.

Principal Results

Several code-mapping examples are developed in an eXtensible Markup Language (XML) environment. The mappings provide the foundation content for an XML document, with the XML structure based on an initial development from the International Council for the Exploration of the Sea (ICES) and the Intergovernmental Oceanographic Commission (IOC) Study Group on the Development of Marine Data Exchange Systems Using XML (SGXML). The mapping examples include code conversions that are developed from eXtensible Stylesheet Language Transformations (XSLT). These conversions provide example algorithms that may be useful when exchanging data and information between database systems.

Significance of Results

System development often attempts to address particular or specific processing needs. Such development typically results in insular systems that complete the required processing with minimal communication with other systems. Often these insular

systems parameterize complex processes or procedures that are explicitly dealt with in external systems. In these cases, cross-communication between systems would enhance system effectiveness.

However, to be effective, cross-communications must establish more than just data transfer. Effective cross-communication implies a shared understanding of the information and context between the separate systems.

In today's database-oriented approach to storage, much of the system specific information will be contained in the form of codes. Since most existing systems were initially developed in isolation, the codes will be system specific. In this case, to obtain cross-communication, a shared understanding of the codes is required.

By utilising new technologies, such as XML, we hope to enhance a system level understanding of the codes by making code manipulation a more open process. This will in turn reduce code-related exchange complications, thus providing more effective cross-communication and understanding.

Future Plans

The code sets used within the LC2IEDM will continue to be investigated. Particular emphasis will be placed on the relationship of the LC2IEDM codes to those code sets used within systems of importance to the Canadian Forces. Only through a complete understanding of the code sets and the relationship between code sets, will true sharing of unambiguous information be attained.

Isenor, Anthony W. 2003. XML Based Manipulation of Codes Exchanged Between Data Systems, DRDC Atlantic TM 2003-132, Defence R&D Canada – Atlantic.

Sommaire

Situation générale

Les données et les informations sont souvent conservées dans une base de données sous forme codée. Les codes, qui sont constitués de lettres, de chiffres ou de mots, constituent un mécanisme pour économiser l'espace dans la base de données. En général, les bases de données autonomes comportent des codes particuliers, qui sont élaborés en fonction du problème spécifique pour lequel la base de données a été conçue, et ils sont fondés sur l'expérience de celui qui modélise les données.

Lorsque des informations sont échangées entre deux bases de données, il se peut que l'information codée dans une base ne soit pas directement transférable dans l'autre base. Des codes semblables mais différents peuvent exister dans les deux bases de données, ou les codes existants qui représentent des variables peuvent être représentés dans des unités différentes. Dans ce cas, une correspondance entre les codes (opération de transcodage) doit être établie.

Cette correspondance associe les codes dans une base de données particulière aux codes dans une autre base de données. L'établissement de correspondances de ce genre est souvent une tâche fastidieuse et laborieuse, nécessitant l'intervention de spécialistes du domaine.

Le Modèle de données pour l'échange d'information de commandement et de contrôle (Terre) (MDEIC2T ou LC2IEDM) constitue une base de données qui intéresse tout particulièrement les chercheurs de la communauté militaire. Ces derniers sont en train d'élaborer ce modèle de données afin de créer un contexte commun entre des systèmes hétérogènes. À ce titre, le jeu de codes complet exploité par le MDEIC2T présente un intérêt tout particulier, étant donné le transcodage requis entre ce modèle et ces systèmes.

Principaux résultats

Plusieurs exemples de transcodage sont fournis dans un environnement XML (eXtensible Markup Language). Ces correspondances de codes ou transcodages constituent la base du contenu d'un document XML, dont la structure est basée sur des travaux préliminaires effectués par le Conseil international pour l'exploration de la mer (CIEM) et le groupe d'étude sur le développement de systèmes d'échange de données maritimes au moyen de XML (SGXML) de la Commission océanographique intergouvernementale (COI). Les exemples de transcodage comprennent des conversions de codes élaborées à partir de XSLT (eXtensible Stylesheet Language Transformations). Ces conversions fournissent des algorithmes d'exemple qui peuvent s'avérer utiles dans les échanges de données et d'informations entre des systèmes de base de données.

Importance des résultats

Dans le développement d'un système, on tente souvent de répondre à des besoins de traitement particuliers. Ces travaux de développement conduisent en général à des systèmes isolés, dans lesquels les traitements requis sont effectués en minimisant les interactions avec d'autres systèmes. Il arrive souvent que ces systèmes isolés paramétrisent des procédures ou des processus complexes qui sont explicitement pris en charge dans des systèmes externes. Dans les situations de ce genre, une intercommunication entre les systèmes améliorerait l'efficacité de ces derniers.

Cependant, pour être efficace, cette intercommunication doit prendre en charge plus que le simple transfert des données. Une intercommunication efficace nécessite une compréhension partagée de l'information et du contexte des deux systèmes.

Aujourd'hui, avec la prévalence de l'approche du stockage orientée vers les bases de données, la plupart des informations propres au système se présentent sous la forme de codes. Comme la plupart des systèmes existants ont été développés au départ isolément, chaque système dispose de ses propres codes spécifiques. Pour établir l'intercommunication, une compréhension partagée des codes est donc requise.

En faisant appel aux nouvelles technologies, notamment XML, nous espérons améliorer la compréhension des codes au niveau du système, en faisant du processus de manipulation des codes un processus plus ouvert. Cela aura à son tour pour effet de réduire les problèmes liés aux échanges de codes, favorisant ainsi la compréhension et l'intercommunication.

Plans pour l'avenir

Les jeux de codes utilisés dans le MDEIC2T vont continuer à faire l'objet d'études. Une attention toute particulière sera portée aux relations qui existent entre les codes de ce modèle et les jeux de codes utilisés dans les systèmes qui sont importants pour les Forces canadiennes. C'est uniquement grâce à une compréhension complète des jeux de codes et des relations qui existent entre eux qu'on pourra réaliser un véritable partage de l'information, sans ambiguïté.

Isenor, Anthony W. 2003. XML Based Manipulation of Codes Exchanged Between Data Systems, DRDC Atlantic TM 2003-132, Defence R&D Canada – Atlantic.

Table of contents

Abstract i

Executive summary iii

Sommaire v

Table of contents vii

List of figures ix

1. Introduction 1

2. Background 4

2.1 Code Usage in LC2IEDM 6

2.2 XML Basics 8

2.3 XSLT Basics 10

3. Code Mapping 13

3.1 Code Conversion 13

3.2 Related but Inexact Code Mappings 18

3.3 Non-Linearity 21

4. Code Mapping in the Canadian Profile Structure 22

4.1 XSLT Parameters 22

4.2 Attributes 23

4.3 data_point, depth_pressure, and previous_value 23

4.4 variable 23

4.5 units 24

4.6 history, quality, and sampling 24

5. Concluding Remarks 25

6. References 26

Annex 1: XSL Code for Brick Manipulation 27

Annex 2: XML Representation of Code Dictionary 32

List of symbols/abbreviations/acronyms/initialisms 33

Distribution list 34

List of figures

Figure 1. Codes in a database may be stored in a domain table as represented above. Categorical variables store the code and the definition. 2

Figure 2. An illustration of data exchange that uses a common exchange structure or format. The central, circular arrow that depicts the flow of data around the circuit represents the exchange structure. This data structure is not linked to a particular in-house database architecture. 4

Figure 3. An illustration of data exchange among common database architectures. Here, data is being exchanged from the in-house database to a hub database. The hub database then forms a communication circuit with the other hub databases. 5

Figure 4. An illustration of data exchange via a centralised database. Here, data is being exchanged from the in-house database to a single instance of a centralised database. All in-house systems communicate with the central database. 6

Figure 5. An example usage of xsl:template. 11

Figure 6. A diagram of the code mapping structure developed by the SGXML [13]. 14

Figure 7. An example XML data structure based on Canadian profile structure [2]. 14

Figure 8. A simplified version of the SGXML code mapping XML structure. Three codes are included in this listing, from three different laboratories. 16

Figure 9. Example XSLT code that converts the XML data document shown in Figure 7 based on the mapping presented in Figure 8. The result of the applied mapping is shown in Figure 10. 17

Figure 10. Results of the application of the code mapping presented in Figure 8 to the XML data document in Figure 7. 18

Figure 11. The code mapping XML document with multipliers inserted to account for unit conversions. Inserted multiplier lines are bold for clarity. The multipliers represent an addition to the SGXML [13] structure. As compared with Figure 8, the unit_of_measure element has also been added. 20

Figure 12. Results of the application of the code mapping presented in Figure 11 to the XML data document in Figure 7. 21

1. Introduction

The storage of data and information in a database takes many forms. In the simplest case, the data are represented as characters, numbers or perhaps files. An example might be a database containing names (character strings), phone numbers (numbers) and photographic images (other files). In this case, the content of the database would be familiar to many people.

However, in some cases the data in a database is not in a familiar form. For example, when repeating information is required throughout the database, the database modeller may introduce abbreviated forms for the repeated information. These abbreviations are a form of coded information.

A *code* may be defined [1] as "a system of signals or of characters used to represent letters or words, or in any way to communicate intelligence". By this definition, codes represent a form of condensed information. The type and content of the condensed information is completely at the discretion of the designer.

Codes may or may not be arbitrary. A simple example of a coded information set is the coding of environmental data types. Here, we use *data type* to represent one category of data. An example of an environmental data type would be temperature. An example code for such a data type might be TEMP. TEMP would be used within the database to represent a specific category of temperature values.

In a database, there are two benefits to code use: 1) compaction of information, and 2) clarity of definitions. To extend the TEMP example, this particular code may mean temperature on the International Temperature Scale 1990 (commonly termed ITS-90), measured in degrees Celsius, and recorded from an electronic thermometer. The usefulness of the database manifests itself by allowing relationships to be created from the definition to the code, and from the code to the data values. Then, only the code is stored with the temperature value, as opposed to storing the full definition with every temperature value. This represents compaction of information. The clarity of definition results in the knowledge that each TEMP value is well defined in terms of scale and recording instrumentation.

Internal to the database, codes are often used within domain tables and categorical variables. In data modelling terminology, *domain tables* contain the allowable content for a particular variable. These tables define the data space or domain of the variable. *Categorical variables* are those variables with allowed content described as categories. The categorical variable is often one column in the domain table. An *enumerated list* refers to the listing of allowed content. An enumerated list has content that is identical to all the categories for a particular categorical variable.

Codes can be powerful within a multi-linguistic environment. This is because codes may be hidden from the user during interactions with the database. In this case, codes are used internally to the software to control program flow. As an example, one could

envisage a database using a set of codes for something as simple as colour. The database could be filled with codes for red, blue, and green. A simple domain table links the code to its definition. In the domain table, the codes would be stored in a variable named "colourcode" and have values as given in Figure 1. The categorical variable "colour" would store the definition for the particular code. In the English version of the database, the domain table content would be as given in Figure 1. In the French version of the database, the codes could remain the same while the definitions in the "colour" variable would change to rouge, bleu, and vert. Smart applications interacting with the database would then use the codes internal to the application, but would use the definition when interacting with the user.

colour	colourcode
green	GR
red	RD
blue	BL

Figure 1. Codes in a database may be stored in a domain table as represented above. Categorical variables store the code and the definition.

In other cases, the actual codes may be provided to the user. This is typically the case with data exchanges, when the user may request data from the database. Extending the temperature example, the user may request all temperature data for a particular region. The data resulting from this request might be presented as columns of data where TEMP is contained in one of those columns. The user would also have access to the definition of TEMP.

In large systems, the codes and definitions may be managed in a system separate from the data. In this case, the code system is referred to as a *dictionary*. Similar to the common language dictionary, a code dictionary evolves to address the topic area of the community. Often, the evolution takes on local characteristics, where codes and definitions are added to meet the local needs of the data processing system. In some cases the local codes are directly linked to the data processing, where processing flow is dictated by the encountered codes.

In these cases, the development of new codes becomes very insular. This does not pose a problem when the organisations are themselves autonomous. However, if the organisations develop a requirement for data and information sharing, the exchange of coded data presents numerous problems.

One solution being sought by many champions of improved data exchange is for all similar organisations to use the same set of codes. In this system, the exchange of data and information would not require code manipulation. However, the system has certain implied requirements. Such a system would necessitate some type of central

authority to maintain the code set. A mechanism would also be required to allow local users the ability to add codes for immediate use. However, such immediate use codes would need to be reviewed by the central authority on some time scale, for possible insertion into the community code set.

There are three key issues that result from the singular code set system. First, the system is totally dependent on the establishment and continuing function of the central authority. Without such an authority, the system collapses. Second, the autonomy of the organisations is slightly compromised, as the established codes may not comply directly with established data models or processing at the local organisation level. In these cases, the local organisation would need to manipulate data structures and processing algorithms to comply with the established codes. Third, if a separate organisation decides not to comply with the standard code set, and this organisation is a substantial contributor to the overall pool of data, then the system breaks down. In this case, the system is forced to recognise the major contributions from the non-compliant organisation. In all three cases, when the system breaks down, the result is a multi-code set situation.

The alternate view of code set development is that each local organisation would maintain their own code set. However, to necessitate data and information sharing, the local code sets must be either transferable or mappable. Here, *mappable* means the code in the provider's code set has an equivalent or near-equivalent code within the receiver's code set.

The mapping is required because often the two code sets are not directly equivalent. A typical example of related but differing codes may involve time stored in seconds in one database and days in another database. Although the conversion is trivial between the two units, a code that encapsulates the unit for time may not create an obvious requirement for conversion. This may result in problems during a data exchange between the two databases.

This paper explores the transfer of coded information between databases with the focus being the transfer of codes that are used in a database. Such code transfers have evolved over recent years, particularly with the introduction of the eXtensible Markup Language (XML). XML provides a framework to develop other languages and is very well suited to transferring the content and structure of databases between different platforms.

The examples presented in this report will use two developed XML languages to define the structures for the data and the code dictionaries. The data structure contains coded information that uses the codes in the data provider's database. Example mappings between the provider's code set and the receiver's code set are created using a map of code equivalents. The mapping is performed using eXtensible Stylesheet Language Transformations (XSLT). XSLT has been developed from XML and is a language designed to restructure and present the content of XML documents.

2. Background

One premise of the networked model is data and information sharing. The details of this sharing are complicated. The structure and content of in-house systems must be taken into account. The developers, engineers and scientists involved in establishing the exchange must all be working together to ensure the needed information is shared in a form that is usable by the other systems within the network. Equally important, the sharing community must recognise those data and information that should not be shared. This excluded data and information may not be required by the community at large, may not be releasable, or may consume large amounts of bandwidth resource.

A model for information sharing may be implemented in many different ways. For example, in one exchange model, a common transport structure is defined (e.g., see [2]). The parties involved then transfer the data using this structure (see Figure 2). In this scenario, the central structure becomes quasi-equivalent to a database. The structure, in the short term, stores the data from one in-house data architecture for delivery to another architecture.

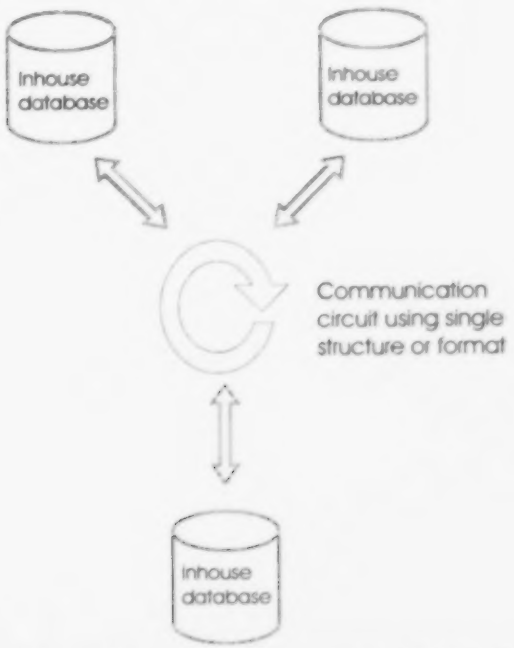


Figure 2 An illustration of data exchange that uses a common exchange structure or format. The central, circular arrow that depicts the flow of data around the circuit represents the exchange structure. This data structure is not linked to a particular in-house database architecture.

An alternate method of exchange would have the central structure of Figure 2 described explicitly within a database. This common database structure would then be linked to the in-house databases, and also linked to other common structures (see Figure 3). In the common database structure, a database layer is added between the in-house database and the central communication system. The common database structure located with a specific provider, then forms one instance of the common database. The party then exchanges information between the in-house database and an instance of the common database. The instance of the common database then communicates the data to the other common databases within the communications network. In this case, the common database forms part of a central hub. This is the model used by the Land Command and Control Information Exchange Data Model (LC2IEDM, or Generic Hub [GH, see 3]).

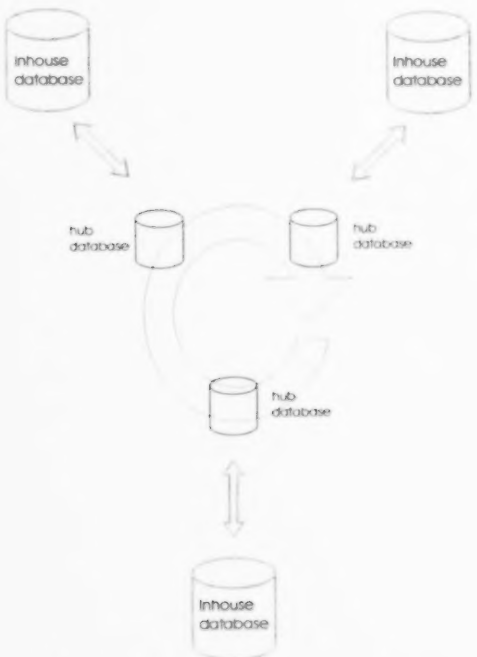


Figure 3. An illustration of data exchange among common database architectures. Here, data is being exchanged from the in-house database to a hub database. The hub database then forms a communication circuit with the other hub databases.

A third scenario collapses the hub database into a single shared database (Figure 4). In this scenario, a centralized database controls all aspects of the data between the systems. This is a more formal, and perhaps more structured, version of Figure 2. This scenario is the present model used by the Maritime Information Shared Technology (MIST, see [4]) data system.

In all scenarios, the local requirement remains the same. The data contained within the in-house databases must be exchanged with the central structure, either the exchange structure or the central database structure. Similarly, the in-house database must be capable of receiving and understanding information from the central structure. This data and information exchange will undoubtedly require code translations in the exchange process.

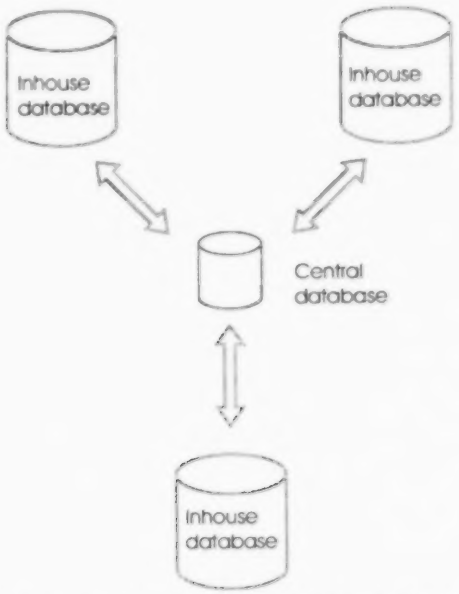


Figure 4. An illustration of data exchange via a centralised database. Here, data is being exchanged from the in-house database to a single instance of a centralised database. All in-house systems communicate with the central database.

2.1 Code Usage in LC2IEDM

Codes are a common occurrence within a database. As an example of code usage within a database system, consider the LC2IEDM.

The LC2IEDM was designed to store all aspects of data required for the logistical and operational battlespace. The data model consists of about 196 tables. The description of the model used in this report is based on version 5.0 documentation (dated 18 March 2002) and the Version 5.3 data model (dated 18 February 2003).

The details of the LC2IEDM presented here do not contribute directly to the code examples that follow in this report. LC2IEDM code usage is included here to

emphasize that codes are an integral and important component of any database model – a component that must be accounted for when exchanging data among systems.

As noted previously, codes are typically an extensive part of any data model. In the LC2IEDM, codes are used in one or more columns in 65% of the tables. This 65% omits those tables using trivial categorical codes such as "YES", "NO", "HUSBND", and "GRDMTH" representing yes, no, husband and grandmother, respectively.

To explore code usage within the LC2IEDM and thus the importance of codes, consider two tables in detail: the control-feature-type and the facility-status.

The table control-feature-type is an object defined as "A non-tangible FEATURE-TYPE of military interest that may be represented as a geometric figure and is associated with the conduct of military operations"[5]. In the data model, this table is named CTRL_FEAT_TYPE. This table consists of four columns one of which is cat_code (e.g., category code). cat_code is defined as "The specific value that represents the class of control-feature-type". cat-code is a six character code, and represents a classification of a feature that is of military interest and one that the military may wish to have under their control (thus the name control-feature).

The enumeration list for cat_code (see [5]) consists of an extensive list of feature types. These types include categorizations such as alternate supply routes (code ALTSPL), air space classifications (codes CLSASP, CLSBSP, CLSCSP, CLSDSP, CLSESP, CLSFSP, and CLSGSP), and lateral routes (code LATR). In total, 264 features (and thus codes) are available for the column cat_code. This example indicates the domain for a single code (thus the name domain table in database terminology). The domain size for the code results in complications for user and software interactions with the data.

The codes stored in the LC2IEDM data model are not explicitly stored in domain tables. The data model stores the allowable content (in this case the codes) in valid lists, which then become part of the structured query language (SQL) commands used to generate the database. There are implications to this storage method. For example, this method requires an update to the data model and generation SQL when a code is added, modified or removed. This minimizes application-level code generation and results in the code set control being placed at the data model level with a central authority (e.g., the committee controlling the data model).

The second table in the LC2IEDM to consider is facility-status. The facility-status table is an object defined as "An OBJECT-ITEM-STATUS that is a record of condition of a specific FACILITY". The table for facility-status, named FAC_STAT consists of 13 columns. However, the interesting point regarding this table is that of the 13 columns, nine store coded information.

The nine code attributes in facility-status are described below. Inside the trailing parentheses is the name of the attribute and the number of items in the enumeration list.

- the class of the facility (category-code - 2),
- the status of a facility destined for demolition (demolition-status-code - 9),
- code indicating if the facility contains mines (mine-presence-code - 3),
- code indicating the presence of an occupation indicator (occupation-program-indicator-code - 3),
- operational status of the facility (operational-status-code - 6),
- code representing the qualification of the operational status (operational-status-qualifier-code - 18),
- code indicating if a facility has been placed in reserve (reserve-indicator-code - 2),
- code indicating security status of the facility (security-status-code - 4), and
- code indicating usage of the facility (usage-status-code - 3)

The list indicates that the number of items in the enumerated list for any one attribute is not large. However, the number of code combinations is over 400,000. This means that the object instance may take one of 400,000 forms. Software interacting with this object will need to interpret a large number of code combinations.

The examples from the LC2IEDM illustrate the volume of codes in a single column and the prominence of code columns in some tables. When dealing with a large volume of codes, problems may result when mapping definitions of codes. In the case of many code columns in a single table, problems may result when a group of codes in one database system represents a smaller (or larger) group of codes in another database system.

2.2 XML Basics

This report will explore a method of code translation using XML. Thus, a brief XML introduction is provided to give the reader enough information to understand the remaining report sections.

XML was developed by the World Wide Web Consortium (W3C) with the release of the specification in 1998 and an updated specification in 2000 [6]. XML is actually a language used to develop other languages. The developed language is thus based on XML, but is not properly named XML.

In the simplest of terms, XML may be used to construct a language using any computer based character set. XML provides various structures that may be used to capture the data. The most basic XML structures are elements and attributes.

An XML *element* is similar to a data object. It may contain other elements or attributes. XML syntax used to identify an element is the angle bracket, < and >. For example, the element named person would be written as

```
<person>
```

The actual text and angle brackets represent the *tag*. To close an XML element, the / is included in the trailing tag. For example,

```
<person>
</person>
```

Alternately, an empty tag may be shortened to be:

```
<person/>
```

To encapsulate another element inside the person element, the syntax would be

```
<person>
  <address>PO Box 1012</address>
</person>
```

Here, the leading spaces on the address element are included for clarity. The content of the address element was also assigned to be "PO Box 1012".

An *attribute* for an element is included within the starting tag. For example, if the address element had an attribute "street" and the name of the street was "Grove", then the syntax would be

```
<person>
  <address street="Grove">PO Box 1012</address>
</person>
```

A *namespace* may also apply to the developed language. Although namespaces will not be dealt with in detail, they do appear in the XSLT elements within this report. A namespace may be considered a specifically named topic area for the developed language. For example, if the developed language for a project was "ABC Language" and the namespace "abc" was declared to represent this language, then the namespace addition would be

```
<abc:person>
  <abc:address street="Grove">PO Box 1012</abc:address>
</abc:person>
```

Finally, in XML terminology, there are well-formed documents and valid documents. *Well-formed* means the start and end tags occur in the proper sequence, similar to the stack first-in-last-out feature. *Valid* means the document agrees with a predefined structure.

There are many more syntactic rules for constructing XML based languages. These rules will not be reviewed here. Those interested are referred to the many on-line resources or published books on the subject (see [7], [8]).

The philosophy behind XML should be noted, and in particular its relationship to codes. An XML based language can be developed from any computer character set. The XML processors deal with this diversity using internal conversions. The processors convert incoming characters into Unicode UTF-16 [9]. In essence, the incoming codes are converted to a common code base. This allows localization of the developed language to suit the local needs, while not imposing global standards on the system.

2.3 XSLT Basics

The examples provided in this report will use Extensible Stylesheet Language Transformations (XSLT) to manipulate the codes contained in an XML file. XSLT is one part of the Extensible Stylesheet Language (XSL), the other parts being Formatting Objects (FO) and XPath. XSLT is a programming language that has been built from XML (see [10]). The XSLT specification is presently at the recommendation stage [11].

XSLT is considered a declarative language, although it does have some characteristics of a procedural language, such as looping instructions. A *declarative* programming language uses the concept of expressions that return values. In *procedural* programming, the equivalent would be statements that modify variables.

There are 36 elements defined in the XSLT specification. These elements provide the manipulation tools that transform the XML input into a specified output form. There are also nine defined XSLT functions and another 27 functions inherited from the XPath specification (described below).

Before proceeding, definitions are required to set the language used throughout this section and the remainder of the report. Here, an *XML document* will be defined as any representation of the XML content. This representation could be in memory, in magnetic storage (e.g., a file on disk), or streamed to an input/output device

A *tree* represents the XML document structure. The tree is a conceptual structure that resembles a physical tree, in that the main root has extending from it branches and leaves. In the case of XML, the branches and leaves are nodes that contain data and structure. A *node* is any element, attribute or content in the XML document. A

node-set is any group of nodes that have a common characteristic. This characteristic may be related to the structure, tag name or content.

Within XSLT, the main concept for transforming the data stream is the template. A *template* is a type of form, into which data from the XML input is placed. All XML data that fit the criteria of the template will be reformed to the rules defined within the template. Templates may be named, or unnamed. If unnamed, then the match attribute of the template defines the template's data space of operation. If named, then the template may be called, analogously to calling functions in a procedural language.

Finally, the *result tree* is the result of combining the input XML document with the XSLT. The result tree is often (but not always) an XML document.

Syntactically in XSLT, a template is defined by the xsl:template element. The xsl:template element describes the structure or form of the required result tree. A template is matched to a particular node-set using another XML based language, XPath. *XPath* is a notation that allows the definition of criteria and the matching of these criteria to node-sets in the XML document [12].

Another important XSLT element is xsl:value-of. The xsl:value-of element provides the ability to output selected node content from the XML document. The xsl:value-of element writes the selected node-set to the result tree.

An xsl:template example is shown in Figure 5. The first line in Figure 5 establishes the content as XSLT. Next, the template matches all elements in the XML document that are "person" with the sub-element "address". In XPath notation, this is represented by "person/address". For all such elements, the result tree is formed with the character string "The persons address is " with the content from the attribute "street" within the address element. The content is obtained using the xsl:value-of element. The XPath notation for selecting the attribute content is "@street". The xsl:text element encapsulates the text to be output to the result tree.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
        <xsl:template match="person/address">
                <xsl:text>The persons address is </xsl:text>
                <xsl:value-of select="@street"/>
        </xsl:template>
</xsl:stylesheet>
```

Figure 5. An example usage of xsl:template.

Other important elements in the XSLT language are xsl:variable, xsl:param and xsl:output. The variable element allows the encapsulation of a data item in a named xsl:variable that can then be referenced by name throughout the template. This is a slight deviation from a pure declarative language, where variables are not allowed.

The xsl:param element allows communication between the operating system and the XSLT parser. xsl:param provides a mechanism to transfer input from the command line to the XSLT program.

The xsl:output element allows the specification of different output formats for the result tree. Two common output formats are text and XML.

3. Code Mapping

It is common for codes to be used throughout database content. The LC2IEDM showed an example of code content within a database. When exchange or sharing of this content is required, some form of code manipulation or mapping will also be required.

The ICES-IOC Study Group on the Development of Marine Data Exchange Systems Using XML (SGXML) is developing a generalized code mapping structure. The primary concern of the SGXML is the exchange of oceanographic data between centres. Such datasets are rich with parameter codes depicting the particular variable. Thus, the first SGXML application of the generalized structure is to the specific case of parameter code mapping.

The mapping structure developed by the SGXML is presented in the SGXML 2002 report [13]. The structure was developed from the recognition that a mapping between parameter codes was necessary to move forward on the seamless sharing of data and information within the community. As well, the SGXML recognised the need for equality between code sets. A pictorial view of the structure is presented in Figure 6.

The mapping shown in Figure 6 uses a form similar to a common language dictionary. A single dictionary will contain many dictionary terms. Each term will in turn contain many definitions where each definition is based on a particular code set. Each code set has a defined name and a code that applies to that dictionary term. Note that there is no order dependency to the definitions - each is considered equal within the dictionary entry.

3.1 Code Conversion

To further the explanation of the code set mapping, consider a situation involving the exchange of a single data value representing elapsed time, between different laboratories. We will share this single value using a highly simplified version of a Canadian ocean profile data structure. The XML data structure is shown in Figure 7. The details of this structure are presented in [2].

The simplified input data file (Figure 7) contains multiple data_set elements and a single data_point element. The data_set element encapsulates and defines multiple datasets within a single document. The data_point element encapsulates the data value and a code representing the parameter. In this example, one elapsed time data point with code ETD has a value 1.

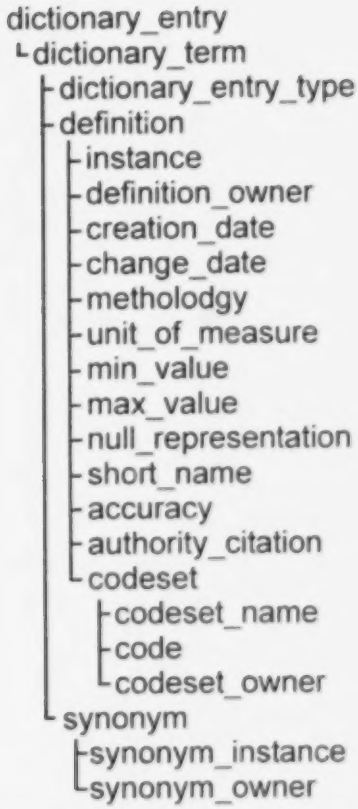


Figure 6. A diagram of the code mapping structure developed by the SGXML [13].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data_collection>
 <data_set>
  <data_set>
   <data_set>
    <data_set>
     <data_point pt_code="ETD">1</data_point>
    </data_set>
   </data_set>
  </data_set>
 </data_set>
</data_collection>
```

Figure 7. An example XML data structure based on Canadian profile structure [2].

To transform this data to the code set used by some other laboratory, we construct a simple mapping using the SGXML structure. The mapping (Figure 8) shows a dictionary containing a single dictionary entry. All other entries have been omitted for clarity. In the more general case, there would be many dictionary entries in the dictionary. Within each dictionary entry there are multiple definitions.

The structure shown in Figure 8 is slightly different from the SGXML structure presented in Figure 6. This is due to modifications to the structure since the publication of the SGXML report [13]. The primary difference involves the promotion of the definition tag to a level equal to dictionary_term. In Figure 6, definition is contained within dictionary_term.

The equivalents among codes are represented by the multiple definitions within the same dictionary entry. The intention here is that the providing and receiving organisations would be included in the definitions. Here, these organisations are denoted Lab-1, Lab-2 and Lab-3 and are considered code set owners.

The transformation of the parameter code in the data file will be carried out using XSLT. The transformation (Figure 9) manipulates the input XML stream contained in Figure 7 and converts the parameter code from one code set to another.

This XSLT document (Figure 9) consists of five main parts, with each part separated by a blank line. The first part is the identifying stylesheet line, output type and input parameters. The parameters are named "input_codes" and "output_codes". These parameters represent the code set owner name. The "input_codes" is the code set owner of the incoming data. Similarly, "output_codes" is the code set owner of the output data stream. In the example, the incoming codes are from the Lab-1 code set, while the outgoing codes are those from the Lab-3 code set.

The second part of the XSLT is a general template that matches all elements, attributes, text and comments in the input document. This template results in all content, except data_point, being copied to the output stream.

The third part is a template for the data_point element. The first section of this template accesses the code-mapping file (dictionary_ex1.xml), and determines the node number of the dictionary entry element that contains the pt_code value within the data_point element.

This third part of the XSLT code also illustrates the priority characteristic of XSLT templates. Note that in the case of the data_point element, a template match is possible for both the general template (denoted by *|@*|text()|comment() match) and the specific template (denoted by the data_point match). In effect, more than one template in the stylesheet matches the pattern of the data_point element. However, XSLT only allows one template to be evaluated for each element. XSLT templates are prioritised, with higher priority templates being evaluated. If an exact match is specified in the template, then this results in a higher priority than the general match indicated by the *|@*|text()|comment(). The result is the evaluation of the data_point template when the data_point element is encountered.

```
<?xml version="1.0"?>
<dictionary>
 <dictionary_entry>
  <dictionary_term>Elapsed Time</dictionary_term>
  <definition>
   <codeset>
        <code>ETD</code>
        <codeset_owner>Lab-1</codeset_owner>
   </codeset>
  </definition>
  <definition>
   <codeset>
       <code>ETS</code>
       <codeset_owner>Lab-2</codeset_owner>
   </codeset>
  </definition>
  <definition>
   <codeset>
       <code>ETM</code>
       <codeset_owner>Lab-3</codeset_owner>
   </codeset>
  </definition>
 </dictionary_entry>
</dictionary>
```

Figure 8. A simplified version of the SGXML code mapping XML structure. Three codes are included in this listing, from three different laboratories.

The fourth part of the XSLT, which is also the second section in the data_point template, accesses the code based on the node number of the dictionary entry and the output_codes code set being requested. An attribute is added to the element, since the copy command only copies the data_point element and content, without the attribute.

The fifth part of the XSLT is the template that determines the node_number for the third part of the style sheet. Having the node number for the dictionary_entry reduces the complexity of the search for the new code. This reduction only applies when multiple dictionary_entry elements are in the map file. The example map file presented here (Figure 8) only contains one dictionary_entry.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="xml" indent="yes"/>
 <xsl:param name="input_codes" select="'Lab-1'"/>
 <xsl:param name="output_codes" select="'Lab-3'"/>

 <xsl:template match="*|@*|text()|comment()">
  <xsl:copy>
   <xsl:apply-templates select="*|@*|text()|comment()"/>
  </xsl:copy>
 </xsl:template>

 <xsl:template match="data_point">
  <xsl:variable name="node_number">
   <xsl:apply-templates
select="document('dictionary_ex1.xml')/dictionary/dictionary_entry/definition/codeset[c
odeset_owner=$input_codes]/code">
    <xsl:with-param name="base_code" select="@pt_code"/>
   </xsl:apply-templates>
  </xsl:variable>

  <xsl:copy>
   <xsl:attribute name="pt_code"><xsl:value-of
select="document('dictionary_ex1.xml')/dictionary/dictionary_entry[number($node_nu
mber)]/definition/codeset[codeset_owner=$output_codes]/code"/></xsl:attribute>
   <xsl:value-of select="."/>
  </xsl:copy>
 </xsl:template>

 <xsl:template match="dictionary/dictionary_entry/definition/codeset/code">
  <xsl:param name="base_code"/>
  <xsl:if test="$base_code = .">
   <xsl:value-of select="1 + count(../../../../preceding-sibling::dictionary_entry)"/>
  </xsl:if>
 </xsl:template>

</xsl:stylesheet>
```

Figure 9. Example XSLT code that converts the XML data document shown in Figure 7 based on the mapping presented in Figure 8. The result of the applied mapping is shown in Figure 10.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data_collection>
 <data_set>
  <data_set>
   <data_set>
    <data_set>
     <data_point pt_code="ETM">1</data_point>
    </data_set>
   </data_set>
  </data_set>
 </data_set>
</data_collection>
```

Figure 10. Results of the application of the code mapping presented in Figure 8 to the XML data document in Figure 7.

3.2 Related but Inexact Code Mappings

The previous example of a code mapping showed how a different code that pertains to the same parameter might be used. This example will consider a slightly more complicated example involving differing units between parameters.

Units may seem innocuous. However, units continually present difficulties for those receiving data. Many serious errors have resulted from incorrect conversion or manipulation of units. The NASA Mars probe [14] is one recent example of a unit error that resulted in complete failure of a mission.

Units present mapping problems often related to applying factors relating one unit to another. As an example, consider a local database storing speed data in the form of metres/second while a central database stores these data in kilometres/hour. In the case of the LC2IEDM, an object item's speed is stored in kilometres/hour (see table OBJ_ITEM_LOC, column speed_rate [5]). Other examples of unit-specific storage of data within the LC2IEDM are found in the tables [5]: PRECIPITATION in column precipitation_rate (millimetres/hour); table WIND column speed_rate in kilometres/hour; and table ATMOSPHERE column temp apparently in degrees Celsius. The documentation for the temp column does not explicitly indicate the units. However, the domain check in the Oracle statements that create the database indicate that the value must be greater than or equal to -274.

Now consider the simple case of a multiplication factor providing the translation between two units. We may extend the previous example by manipulating the elapsed time value between the units of day, minute and second. However, now the factor must be specified within the code mapping. To include the factor, we add an XML element named multiplier as shown in Figure 11. This element is an addition to the

SGXML [13] structure. The multiplier element also contains an attribute pt_link. This attribute represents a point link to another codeset (Note: the single word *codeset* refers to the XML tag, while *code set* refers to a complete group of codes used by a particular organisation.). The pt_link contains the name of the linked codeset.

The multiplication factor and the pt_link attribute represent the conversion from the code set referenced in the pt_link attribute to the codeset containing the multiplier. As an example, the conversion from minute to seconds is a factor of 60, or 60 times a value expressed in minutes results in the value expressed in seconds. In the XML document in the Lab-2 codeset, this relationship is expressed as:

<multiplier pt_link="Lab-3">60</multiplier>

This can be read as, "the Lab-3 code multiplied by 60 results in the Lab-2 code". The Lab-2 code corresponds to a unit of second, while the Lab-3 code corresponds to a unit of minute.

Also note that the multipliers are present in all codesets and relate to all other codesets. It is not absolutely necessary that this relationship to all other codesets exist.

To implement this factoring for unit conversion, one line of the XSLT requires modification. In particular, in the fourth section of Figure 9, the line that reads

<xsl:value-of select="."/>

must be changed to

<xsl:value-of select="self::node() *
document('dictionary_ex2.xml')/dictionary/dictionary_entry[number($node_number)]/
definition/codeset[codeset_owner=$output_codes]/multiplier[@pt_link=$input_codes]
"/>

Applying the revised XSLT to the input presented in Figure 7 using the conversion for the code set named Lab-3, we obtain the results in Figure 12.

The revised line is a representation of the implemented XSLT algorithm. In this line, the node_number corresponds to the dictionary_entry for the input code. Using the node_number, the output codeset_owner is identified and the multiplier corresponding to the input code is found. This multiplier is then used to convert the input value to the corresponding unit of the output code.

```
<?xml version="1.0"?>
<dictionary>
 <dictionary_entry>
  <dictionary_term>Elapsed Time</dictionary_term>
  <definition>
   <unit_of_measure>day</unit_of_measure>
   <codeset>
    <code>ETD</code>
    <codeset_owner>Lab-1</codeset_owner>
    <multiplier pt_link="Lab-2">0.000011574</multiplier>
    <multiplier pt_link="Lab-3">0.000694444</multiplier>
   </codeset>
  </definition>
  <definition>
   <unit_of_measure>second</unit_of_measure>
   <codeset>
    <code>ETS</code>
    <codeset_owner>Lab-2</codeset_owner>
    <multiplier pt_link="Lab-1">86400</multiplier>
    <multiplier pt_link="Lab-3">60</multiplier>
   </codeset>
  </definition>
  <definition>
   <unit_of_measure>minutes</unit_of_measure>
   <codeset>
    <code>ETM</code>
    <codeset_owner>Lab-3</codeset_owner>
    <multiplier pt_link="Lab-1">1440</multiplier>
    <multiplier pt_link="Lab-2">0.01666667</multiplier>
   </codeset>
  </definition>
 </dictionary_entry>
</dictionary>
```

Figure 11. The code mapping XML document with multipliers inserted to account for unit conversions. Inserted multiplier lines are bold for clarity. The multipliers represent an addition to the SGXML [13] structure. As compared with Figure 8, the unit_of_measure element has also been added.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data_collection>
 <data_set>
  <data_set>
   <data_set>
    <data_set>
     <data_point pt_code="ETM">1440</data_point>
    </data_set>
   </data_set>
  </data_set>
 </data_set>
</data_collection>
```

Figure 12. Results of the application of the code mapping presented in Figure 11 to the XML data document in Figure 7.

Figure 12 shows that the initial ETD (elapsed time in days) value of 1 has been converted to a code of ETM (elapsed time in minutes) with a value of 1440. The multiplication by 1440 is based on the multiplier for the Lab-1 to Lab-3 conversion (Figure 11).

The conversion in this example may also be extended to include an offset term. The offset term would also represent an addition to the SGXML code structure.

3.3 Non-Linearity

The above shows a very simple example of a linear operation using XSLT. Non-linear operations are also possible in a declarative language, but are slightly different as compared with a procedural language. In a procedural language, a non-linearity may be represented by a single line of code, or perhaps by recursively executed statements (e.g., using a FOR loop). In a declarative language such as XSLT, non-linear operations are represented as recursive calls to named templates. Thus, if the non-linearity can be represented as a converging series (e.g., Taylor series expansion), then XSLT may be used to represent the non-linear function. In this case, the recursive calls continue until a defined tolerance is attained between successive results from the calculation. In this way, XSLT may be used to represent trigonometric functions, square roots, etc. [15].

4. Code Mapping in the Canadian Profile Structure

As indicated in Section 2, the exchange of data and information may be represented in many forms. Figure 2 indicated an exchange form that utilized a common transport structure for the exchange. This model has been followed in a Canadian effort [2] to define a transport mechanism for oceanographic profile data at the national and international [13] levels using XML.

The Canadian profile XML structure [2] was presented in a very brief form in Figure 7. In general terms, the structure consists of encapsulated data_set elements that define the natural hierarchy found in oceanographic profile data. Metadata may be included at various levels to support the data.

The complete ocean profile XML structure is considerably more complicated than presented in Figure 7. The complete structure is comprised of about 20 pure elements and another five compound elements. The reader is referred to a lengthy discussion of the element types as presented in [2].

Codes in the structure are evident in many of the XML elements, specifically as attribute content. The transformation of the coded information that is contained within the structure must not only deal with the codes, but also must account for unit changes impacting data values. Thus, considerable knowledge on the structure detail must be incorporated into the algorithm. This results in a slightly more detailed algorithm as compared to Figure 9.

The XSLT algorithm to convert all codes between institutes using the Canadian XML structure for point data is given in Annex 1. In this section, the conversion algorithm in discussed. Annex 1 should be referred to throughout the following sections. As well, the form of the XML dictionary representation is presented in Annex 2.

4.1 XSLT Parameters

The input parameters for the XSLT converter are again named input_codes, output_codes and map_document. The input_codes declares the name of the code set used in the mapping file for those codes in the input XML document. Similarly, the output_codes declares the name of the code set required for the result tree. The map_document parameter declares the name of the XML file containing the code mapping.

The output_codes identifies the codes required in the result tree. This code set name is also placed in the dictionary_name sub-element of the dictionary element. This element and sub-element are described in [2].

4.2 Attributes

A template, matching all encountered attributes, copies all attributes and content to the result tree. Only two attributes are dealt with in a special way: pt_code and stored_units. The content of pt_code is changed to be the output code for the requested output code set. Similarly, the units corresponding to the output code are stored in the attribute stored_units.

4.3 data_point, depth_pressure, and previous_value

These three elements are manipulated in a similar fashion. For each, the multiplier found in the code mapping XML document for the required code conversion is used as a multiplier for the content value. The pt_code attribute value for each element is changed to the required output code.

4.4 variable

The variable element contains several unique sub-elements and must therefore be dealt with separately. When the variable element is encountered, a separate template is applied that steps through all sub-elements in the variable element. The code multiplier is applied to the content of the following variable element sub-elements:

- accuracy,
- below_detection,
- maximum_value, and
- minimum_value.

The multiplier contained in the XML map document therefore changes the content of these four sub-elements. All other sub-elements of the variable element are copied without alteration to the result tree.

4.5 units

The units element contains attributes and sub-elements that must be manipulated. In terms of the attributes, the content from stored_units in the input document is first copied to the attribute received_units. In this way, the receiving organisation maintains a record of the units originally received from the data provider. Note that if the input XML document contains the received_units attribute, any content in this attribute will be lost.

The stored_units attribute will be converted to the unit specified by the output code. This conversion takes place when the attribute is encountered as opposed to when the units element is encountered (see Attribute section above).

The conversion sub-element is used to store the multiplier used in the conversion. Any initial content in the conversion sub-element is lost.

4.6 history, quality, and sampling

These three elements are copied to the output stream with the only change being the code specified in the pt_code attribute. The code change takes place in the template matching all attributes.

5. Concluding Remarks

In terms of software and data exchanges between systems, the algorithms and examples presented here are somewhat trivial. However, the problem being addressed by the algorithms and software is far from trivial. The mapping of code sets between systems is a major problem in data and information exchange. This is due in part to the autonomous nature of database development within organisations.

The autonomous developments have resulted in many code sets that are particular to the local database and processing systems. These local systems work well in isolation. However, when these local systems are joined in a data and information exchange capacity, the code sets used within the systems become important.

The mapping of codes from one code set to another is a tedious task. Once completed, the user then requires a method and algorithm for converting the codes. This report outlined one such method using XML and XSLT. A data file was constructed using XML. The code set was also represented in an XML-based language. The codes in the data file were then converted using the code set XML file and an algorithm implemented in XSLT.

The results indicate that XSLT and XML are a viable option for translating codes in XML documents. However, the difficulty in any such translation is the code mapping. The number of codes contained in any one database system may be large and the mapping between two such systems will be a complicated and time consuming task. However, for the exchange of unambiguous data between the systems, a complete and accurate mapping is essential.

6. References

1. Funk and Wagnalls New Standard Dictionary of the English Language, New York, 1950.

2. Isenor, Anthony W., J. Robert Keeley and Joe Linguanti. 2003. Developing an eXtensible Markup Language (XML) Application for DFO Marine Data Exchange via the Web, DRDC Atlantic ECR 2003-025, Defence R&D Canada – Atlantic.

3. The Land C2 Information Exchange Data Model, Data and Procedures Working Group, Working Paper 5-5, Draft 5.3, Greding Germany.

4. CDS – Coalition Data Server, Programmer's Reference Guide, September 26, 2002, Version 1.0, Naval Undersea Warfare Center Division, Newport.

5. The Generic Hub 5 Data Model – Annexes, Baseline 1.0 Draft 0.9, 30 July 2002.

6. Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, see http://www.w3.org/TR/REC-xml

7. The Annotated XML Specification at http://www.xml.com/axml/testaxml.htm

8. Walsh, Norman. A Technical Introduction to XML, http://www.xml.com/lpt/a/98/10/guide0.html

9. Unicode Home Page at http://www.unicode.org/

10. Cagle, Kurt, Michael Corning, Jason Diamond, Teun Duynstee, Oli Gauti Gudmundsson, Michael Mason, Jonathan Pinnock, Paul Spencer, Jeff Tang, and Andrew Watt. 2001. Professional XSL, Wrox Press Ltd.

11. XSL Transformations (XSLT), Version 1.0, W3C Recommendation 16 November 1999, see http://www.w3.org/TR/xslt

12. XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999, see http://www.w3.org/TR/xpath

13. SGXML, 2002. Report of the ICES-IOC Study Group on the Development of Marine Data Exchange Systems Using XML, Helsinki, Finland, 15–16 April 2002, ICES CM 2002/C:12.

14. Isbell, Douglas, Mary Hardin, Joan Underwood. Mars Climate Orbiter Team Finds Likely Cause of Loss, Press Release at http://mars.jpl.nasa.gov/msp98/news/mco990930.html

15. Novatchev, Dimitre. FXSL -- the Functional Programming Library for XSLT, http://fxsl.sourceforge.net/, November, 2001.

Annex 1: XSL Code for Brick Manipulation

The following XSL converts the codes contained in the XML-based structure presented in [2], using a code mapping between the Institutes involved in the data exchange. The XSL was developed specifically for the data structure defined in [2]. For information on XSL, see [10].

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
 <xsl:output method="xml" indent="no"/>

 <!--
Features of this code:
  When the (input code, code set) pair exists in the mapping
  document, but a corresponding (output code, code set) pair
  does not exist, the output pt_code is empty and NaN
  replaces the data value.

  When the (input code, code set) does not exist in the mapping
  document, the original pt_code and value are output.

  When the (input code, code set) exists and the (output code,
  code set) exists, the output pt_code is the code from the
  output code set with the appropriate multiplier applied to
  the data value.

  This code was tested using James Clark's xt program.

                                    Anthony W. Isenor April 2003
-->

 <xsl:param name="input_codes" select="'BIO-Canada'"/>
 <xsl:param name="output_codes" select="'IOS-Canada'"/>
 <xsl:param name="map_document"
select="'canadian_code_mapping.xml'"/>

 <xsl:template match="*|@*|text()|comment()">
  <xsl:copy>
   <xsl:apply-templates select="*|@*|text()|comment()"/>
  </xsl:copy>
 </xsl:template>

<!--Set dictionary name to output parameter code.-->
 <xsl:template match="dictionary_name">
  <xsl:element name="dictionary_name">
   <xsl:value-of select="$output_codes"/>
  </xsl:element>
 </xsl:template>
```

```
 <!--       All occurences of pt_code:

data_point|depth_pressure|history|previous_value|quality|sampling|uni
ts|variable

            Change the content value:
data_point|depth_pressure|previous_value
            Change subelement content value: variable|units
            Change pt_code: data_point|history|quality|sampling|units
-->

<!--Call the convert template for all these bricks.-->
 <xsl:template
match="data_point|depth_pressure|history|previous_value|quality|sampl
ing|units|variable">
  <xsl:call-template name="convert"/>
 </xsl:template>

 <xsl:template name="convert">

<!--Find the node number in the map document that contains the code
found in pt_code.  Note that we only search the input_codes.-->
  <xsl:variable name="node_number">
   <xsl:apply-templates
select="document($map_document)/dictionary/dictionary_entry/definitio
n[definition_owner=$input_codes]/codeset/code">
    <xsl:with-param name="base_code" select="@pt_code"/>
   </xsl:apply-templates>
  </xsl:variable>

  <xsl:choose>
   <xsl:when test="number($node_number)">
    <xsl:copy>

    <!--Copy all attributes to the result tree-->
     <xsl:apply-templates select="@*">
      <xsl:with-param name="node_number3" select="$node_number"/>
     </xsl:apply-templates>
     <xsl:choose>
      <xsl:when test="local-name() = 'data_point'">
       <xsl:value-of select="self::node() *
document($map_document)/dictionary/dictionary_entry[number($node_numb
er)]/definition[definition_owner=$output_codes]/codeset/multiplier[@p
t_link=$input_codes]"/>
      </xsl:when>

      <xsl:when test="local-name() = 'depth_pressure'">
       <xsl:value-of select="self::node() *
document($map_document)/dictionary/dictionary_entry[number($node_numb
er)]/definition[definition_owner=$output_codes]/codeset/multiplier[@p
t_link=$input_codes]"/>
      </xsl:when>
```

```
      <xsl:when test="local-name() = 'previous_value'">
       <xsl:value-of select="self::node() *
document($map_document)/dictionary/dictionary_entry[number($node_numb
er)]/definition[definition_owner=$output_codes]/codeset/multiplier[@p
t_link=$input_codes]"/>
      </xsl:when>

      <xsl:when test="local-name() = 'variable'">
       <xsl:apply-templates select="*" mode="var">
        <xsl:with-param name="node_number2" select="$node_number"/>
       </xsl:apply-templates>
      </xsl:when>

      <xsl:when test="local-name() = 'units'">
       <xsl:attribute name="received_units">
        <xsl:value-of select="@stored_units"/>
       </xsl:attribute>
       <xsl:text>&#xA;</xsl:text>
       <xsl:element name="conversion">
        <xsl:value-of
select="document($map_document)/dictionary/dictionary_entry[number($n
ode_number)]/definition[definition_owner=$output_codes]/codeset/multi
plier[@pt_link=$input_codes]"/>
       </xsl:element>
       <xsl:text>&#xA;</xsl:text>
      </xsl:when>

      <xsl:otherwise>
       <xsl:copy-of select="./*"/>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:copy>
   </xsl:when>

   <xsl:otherwise>
    <xsl:copy-of select="."/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>

<!--This template deals with all the sub-elements of the variable
brick.-->
 <xsl:template match="*" mode="var">
  <xsl:param name="node_number2"/>
  <xsl:choose>
   <xsl:when test="local-name() = 'accuracy'">
    <xsl:copy>
     <xsl:value-of select="self::node() *
document($map_document)/dictionary/dictionary_entry[number($node_numb
er2)]/definition[definition_owner=$output_codes]/codeset/multiplier[@
pt_link=$input_codes]"/>
    </xsl:copy>
    <xsl:text>&#xA;</xsl:text>
   </xsl:when>
```

```
   <xsl:when test="local-name() = 'below_detection'">
    <xsl:copy>
     <xsl:value-of select="self::node() *
document($map_document)/dictionary/dictionary_entry[number($node_numb
er2)]/definition[definition_owner=$output_codes]/codeset/multiplier[@
pt_link=$input_codes]"/>
    </xsl:copy>
    <xsl:text>&#xA;</xsl:text>
   </xsl:when>

   <xsl:when test="local-name() = 'maximum_value'">
    <xsl:copy>
     <xsl:value-of select="self::node() *
document($map_document)/dictionary/dictionary_entry[number($node_numb
er2)]/definition[definition_owner=$output_codes]/codeset/multiplier[@
pt_link=$input_codes]"/>
    </xsl:copy>
    <xsl:text>&#xA;</xsl:text>
   </xsl:when>

   <xsl:when test="local-name() = 'minimum_value'">
    <xsl:copy>
     <xsl:value-of select="self::node() *
document($map_document)/dictionary/dictionary_entry[number($node_numb
er2)]/definition[definition_owner=$output_codes]/codeset/multiplier[@
pt_link=$input_codes]"/>
    </xsl:copy>
    <xsl:text>&#xA;</xsl:text>
   </xsl:when>

   <xsl:otherwise>
    <xsl:copy-of select="."/>
    <xsl:text>&#xA;</xsl:text>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>


 <xsl:template match="@*">
  <xsl:param name="node_number3"/>
  <xsl:attribute name="{name()}">
   <xsl:choose>

    <xsl:when test="local-name() = 'pt_code'">
     <xsl:value-of
select="document($map_document)/dictionary/dictionary_entry[number($n
ode_number3)]/definition[definition_owner=$output_codes]/codeset/code
"/>
    </xsl:when>

    <xsl:when test="local-name() = 'stored_units'">
     <xsl:value-of
select="document($map_document)/dictionary/dictionary_entry[number($n
```

```
ode_number3)]/definition[definition_owner=$output_codes]/unit_of_meas
ure"/>
    </xsl:when>

    <xsl:otherwise>
     <xsl:value-of select="."/>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:attribute>
 </xsl:template>

<!--base_code is the input code within the pt_code attribute of the
brick.
The map file is searched until the base_code is found in the map
file.
Then count the number of dictionary_entry to determine the node
number.-->
 <xsl:template
match="dictionary/dictionary_entry/definition/codeset/code">
  <xsl:param name="base_code"/>
  <xsl:if test="$base_code = .">
   <xsl:value-of select="1 + count(../../.././preceding-
sibling::dictionary_entry)"/>
  </xsl:if>
 </xsl:template>
</xsl:stylesheet>
```

Annex 2: XML Representation of Code Dictionary

This Annex contains a single entry from the XML dictionary representation used in the code mapping. This representation, with the XML-based profile data structure [2], can be used with the XSL code (Annex 1) to perform the code mapping.

```
<?xml version="1.0" ?>
<dictionary>
   <dictionary_owner>Canadian DFO Labs - MEDS, IOS and
BIO</dictionary_owner>
   <dictionary_citation>Canadian DFO Code Mapping, XML Version,
February 2003.</dictionary_citation>
   <dictionary_description>This code mapping provides linkages
between ocean data codes used at MEDS, IOS and BIO
(OSD).</dictionary_description>
   <dictionary_entry>
      <dictionary_term>Ammonium (NH4-N)
content</dictionary_term>
      <role>parameter</role>
      <definition>
         <definition_owner>MEDS-Canada</definition_owner>
         <unit_of_measure>mmol/m**3</unit_of_measure>
         <codeset>
            <code>AMON</code>
            <codeset_owner>MEDS-Canada</codeset_owner>
            <multiplier pt_link="IOS-Canada">1</multiplier>
            <multiplier pt_link="BIO-Canada">1</multiplier>
         </codeset>
      </definition>
      <definition>
         <definition_owner>IOS-Canada</definition_owner>
         <unit_of_measure>umol/L</unit_of_measure>
         <codeset>
            <code>Ammonium</code>
            <codeset_owner>IOS-Canada</codeset_owner>
            <multiplier pt_link="MEDS-Canada">1</multiplier>
            <multiplier pt_link="BIO-Canada">1</multiplier>
         </codeset>
      </definition>
      <definition>
         <definition_owner>BIO-Canada</definition_owner>
         <unit_of_measure>mmol/m**3</unit_of_measure>
         <codeset>
            <code>AMON</code>
            <codeset_owner>BIO-Canada</codeset_owner>
            <multiplier pt_link="MEDS-Canada">1</multiplier>
            <multiplier pt_link="IOS-Canada">1</multiplier>
         </codeset>
      </definition>
   </dictionary_entry>
</dictionary>
```

List of symbols/abbreviations/acronyms/initialisms

DND	Department of National Defence
DRDC	Defence R&D Canada
FO	Formatting Objects
GH	Generic Hub
ICES	International Council for the Exploration of the Sea
IOC	Intergovernmental Oceanographic Commission
ITS	International Temperature Scale
LC2IEDM	Land Command and Control Information Exchange Data Model
MIST	Maritime Information Shared Technology
SGXML	ICES-IOC Study Group on the Development of Marine Data Exchange Systems Using XML
SQL	Structured Query Language
UTF	Unicode Transformation Format
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformation

Distribution list

LIST PART 1:

2	DRDC ATLANTIC LIBRARY FILE COPIES
3	DRDC ATLANTIC LIBRARY (SPARES)
1	W. A. ROGER
1	M. E. LEFRANÇOIS
1	G. R. MELLEMA
1	J. S. KENNEDY
1	AUTHOR
10	TOTAL LIST PART 1

LIST PART 2: DISTRIBUTED BY DRDKIM 3

1 NDHQ/ ADMS&T/ DRDKIM 3

1 Pierre Clement
Marine Environmental Sciences Division
Bedford Institute of Oceanography
PO Box 1006
Dartmouth, N.S.
B2Y 4A2

1 Donald W. Collins
U.S. National Oceanographic Data Center
1315 East West Highway, 4th Floor,
Silver Spring MD, 20910,
USA

1 Dr. Harry Dooley
International Council for the Exploration of the Sea (ICES)
2-4 Palægade
DK-1261
Copenhagen K
Denmark

1 J. Robert Keeley
Marine Environmental Data Service
Department of Fisheries and Oceans
W12082 - 200 Kent Street
Ottawa, Ontario Canada
K1A 0E6

1 Joe Linguanti
Institute of Ocean Sciences
PO Box 6000, 9860 West Saanich Road
Sidney, B.C.
V8L 4B2

1 Dr. Roy Lowry
British Oceanographic Data Center,
Proudman Oceanographic Laboratory,
Bidston Observatory, Prenton,
Merseyside, CH43 7RA,
United Kingdom

7 TOTAL LIST PART 2

17 TOTAL COPIES REQUIRED

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – Atlantic
PO Box 1012
Dartmouth, Nova Scotia
B2Y 3Z7

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

XML Based Manipulation of Codes Exchanged Between Data Systems

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Anthony W. Isenor

5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information Include Annexes, Appendices, etc).	6b. NO. OF REFS (total cited in document)
August 2003	35	15

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

TECHNICAL MEMORANDUM

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

Defence R&D Canada – Atlantic
PO Box 1012
Dartmouth, NS, Canada B2Y 3Z7

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant).	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Atlantic TM 2003–132	

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(x) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Codes are an abbreviated form of information that are often found in many databases. Codes provide the database with a compaction of information and concise definition of data values. For most databases, the codes are developed specifically for the local database and processing system. In a data-sharing environment, the specific code development results in different database systems referring to similar data using different codes. The conversion of codes is required when these systems share data. The eXtensible Markup Language (XML) is used to construct a code mapping between two systems and eXtensible Stylesheet Language Transformations (XSLT) are used to convert the codes in the XML documents.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

XML
XSLT
code set mapping
database
LC2IEDM
Land Command and Control Information Exchange Data Model
GH
Generic Hub